

आपकी
सफलता
का साथी



DATA STRUCTURES & ALGORITHMS

COMPLETE STUDY MATERIAL

FOR COMPLETE UNDERSTANDING & EXAM SUCCESS



EASY
LANGUAGE



MCQs WITH
ANSWERS



CONCEPTS WITH
EXAMPLES



DIAGRAMS &
TABLES



EXAM
FOCUSED



COMPLETE
COVERAGE



FOR
RSSB BCI EXAM

CREATED BY

MARUDHARA EXAM



www.marudharaexam.in



मरुधरा एग्जाम

राजस्थान के हर छात्र का
भरोसेमंद साथी



विश्वास आपका
साथ हमारा

RAJASTHAN'S
NO.1
EXAM PLATFORM



www.marudhraexam.in



★ एक ही प्लेटफॉर्म पर सभी सुविधाएं ★



MOCK TEST

विषय अनुसार टेस्ट दें, अपनी
तैयारी को परखें और रैंक प्राप्त करें।



DAILY SUJAS

रोजाना सुजानसर PDFs डाउनलोड
करें, करेंट अफेयर्स रहें अपडेट।



IMPORTANT NOTES

हाथ से लिखे नोट्स और विषय
अनुसार अध्ययन सामग्री।



PDF DOWNLOADS

RPSC, RSSB और अन्य परीक्षाओं
के लिए PDF सामग्री डाउनलोड करें।



QUIZ

रोजाना Quiz खेलें और
अपनी जनरल नॉलेज को
बेहतर बनाएं।



OMR CHECK

अपनी OMR शीट अपलोड करें
और तुरंत रिजल्ट प्राप्त करें।



RESULT PORTAL

अपना रिजल्ट देखें, स्कोरकार्ड
डाउनलोड करें और एनालिसिस करें।



RANK SYSTEM

ऑल इंडिया रैंक, कैटेगरी रैंक और
परफॉर्मंस एनालिसिस देखें।



EMAIL RESULT

अपना पूरा रिजल्ट PDF के रूप में
ईमेल पर प्राप्त करें।



PREMIUM MOCKS

कुछ प्रीमियम टेस्ट केवल ₹11 में
अनलॉक करें और बेहतर तैयारी करें।



MOBILE FRIENDLY

मोबाइल, टैबलेट और कंप्यूटर -
सभी पर शानदार अनुभव।



100% SECURE

आपका डेटा सुरक्षित है, विश्वसनीय
और तेज प्लेटफॉर्म।

- ✓ RPSC, RSSB, CET, VDO, BCI, 3rd Grade Teacher और सभी प्रतियोगी परीक्षाओं के लिए
- ✓ नियमित अपडेट और नई सामग्री
- ✓ बेहतर तैयारी, बेहतर परिणाम
- ✓ आपकी सफलता, हमारा लक्ष्य

आज ही विजिट करें

www.marudhraexam.in

— और अपनी तैयारी को दें नई उड़ान! —

सपना आपका
सफलता
हमारी
जिम्मेदारी



हजारों छात्रों का भरोसा
हर दिन बढ़ता परिवार



कभी भी, कहीं भी
हमेशा आपके साथ



www.marudhraexam.in



Join us for Regular Updates



Trusted by Thousands
of Students Across Rajasthan

1. INTRODUCTION

Data Structures are ways of organizing and storing data in computer memory so that it can be used efficiently.

Algorithms are step-by-step procedures or instructions to solve a problem.

Good data structures + Good algorithms = Efficient Programs

2. ABSTRACT DATA TYPES (ADT)

ADT is a logical / mathematical model that defines a data type along with operations that can be performed on it.

Example → Stack ADT

Operations:

- push()
- pop()
- peek()/top()
- isEmpty()

TOP →

50
40
30
20
10

push → insert

pop → delete

3. ARRAYS AS DATA STRUCTURES

- Array is a collection of similar data elements stored at contiguous memory locations.
- Access time of any element is $O(1)$.

Example →

0	1	2	3	4
10	20	30	40	50

Declaration (C/C++):

```
int arr[5];
```

Index starts from 0.

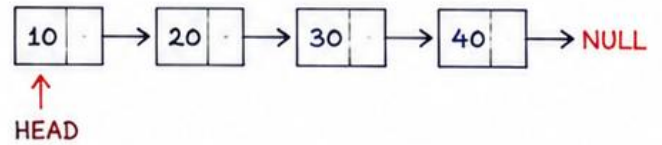


4. LINKED LIST

- Linked List is a **linear** data structure.
- Elements (Nodes) are **not** stored in contiguous memory locations.
- Each node contains **data** and **address** of next node.

Types → Singly Linked List, Doubly Linked List, Circular Linked List

Example (Singly Linked List)



Node Structure (C/C++):

```
struct Node {  
    int data;  
    struct Node* next;  
};
```

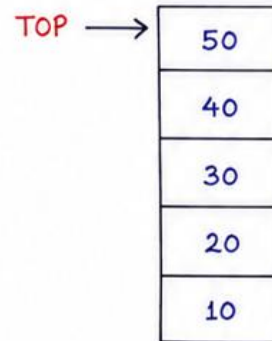
5. STACK

- Stack is a **linear** data structure.
- Works on **LIFO** (Last In First Out) principle.
- Insertion and deletion take place at the same end called **TOP**.
- Used in: Function calls, Expression evaluation, Backtracking, Undo operations, etc.

Stack in C (using array):

```
int stack[MAX], top = -1;  
push(x) → stack[++top] = x;  
pop() → return stack[top--];
```

Stack Representation



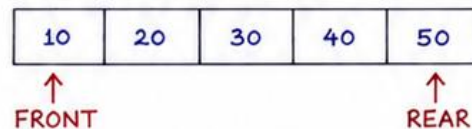
Operations:

- push()
- pop()
- peek()/top()
- isEmpty()
- isFull()

6. QUEUE

- Queue is a **linear** data structure.
- Works on **FIFO** (First In First Out) principle.
- Insertion is done at **REAR** and deletion is done from **FRONT**.
- Used in: CPU Scheduling, Printer Queue, Breadth First Search, etc.

Queue Representation



Operations:

- enqueue()
- dequeue()
- front()/peek()
- isEmpty()
- isFull()

Queue in C (using array):

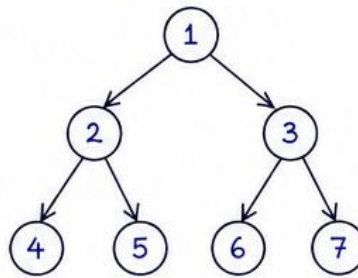
```
int queue[MAX], front = -1,  
    rear = -1;  
enqueue(x) →  
    queue[++rear] = x;  
dequeue() →  
    return queue[++front];
```



7. BINARY TREES

- A tree in which each node can have **at most two children**.
- Children are called **Left child** and **Right child**.
- The topmost node is called **Root**.
- Nodes with no children are called **Leaf nodes**.

Example :



Types of Binary Trees :

- Full Binary Tree
- Complete Binary Tree
- Perfect Binary Tree
- Balanced Binary Tree
- Skewed Binary Tree

Node Structure (C/C++):

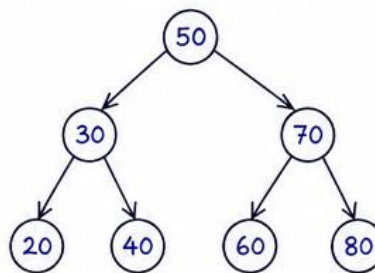
```

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};
    
```

8. BINARY SEARCH TREE (BST)

- BST is a binary tree with ordering property.
- For any node N:
 - All keys in left subtree of N are **smaller** than key in N.
 - All keys in right subtree of N are **greater** than key in N.
- Inorder traversal of BST gives **sorted order**.
- Used in searching, sorting, dynamic sets, etc.

Example :



Node Structure (C/C++):

```

struct Node {
    int data;
    struct Node *left, *right;
};
    
```

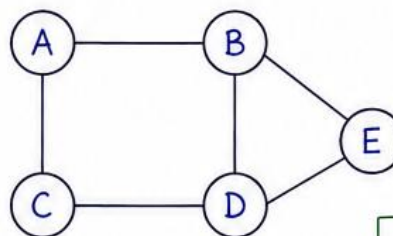
Basic Operations :

- Search
- Insert
- Delete
- Find Min/Max
- Inorder Traversal
- Preorder Traversal
- Postorder Traversal

9. GRAPHS

- Graph is a **non-linear** data structure.
- It consists of a set of **vertices** (or nodes) and a set of **edges**.
- Can be **directed** or **undirected**.
- Can be **weighted** or **unweighted**.
- Used in networks, routing, social networks, scheduling, etc.

Example Graph (Undirected)



Graph Representations :

1. Adjacency Matrix
2. Adjacency List

Basic Operations / Algorithms :

- BFS, DFS, Shortest Path, MST, Topological Sort, etc.

Adjacency List Example:

```

A : B → C
B : A → D → E
C : A → D
D : C → B → E
E : B → D
    
```



10. SORTING ALGORITHMS

- Sorting means arranging elements in a particular order (Ascending or Descending).
- Sorting is a fundamental operation in many applications.
- Improves search efficiency.
- **Common Sorting Algorithms:**

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort

Example (Ascending Order)

Unsorted Array :

64	25	12	22	11
----	----	----	----	----



Sorted Array :

11	12	22	25	64
----	----	----	----	----

Time Complexity Summary :

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

11. SEARCHING ALGORITHMS

- Searching is the process of finding an element in a collection.
- Returns the position or presence of the element.
- **Common Searching Algorithms:**

1. Linear Search
2. Binary Search

(i) Linear Search

- Checks each element one by one.
- Works on **unsorted** arrays.
- Time Complexity :

$O(n)$

Example :

Array :

10	25	30	45	60
----	----	----	----	----

Key = 30

Comparisons : 10 → 25 → 30 (Found)

Position = 3 (Indexing from 0)

(ii) Binary Search

- Works on **sorted** arrays.
- Repeatedly divides the array into halves.
- Time Complexity :

$O(\log n)$

Example :

Sorted Array :

10	20	30	40	50	60	70
----	----	----	----	----	----	----

Key = 40

Steps :

Mid = 30 → Key > 30 → Right Half

Mid = 50 → Key < 50 → Left Half

Mid = 40 → Found

Position = 3

12. HASHING

- Hashing is a technique to map a key to an index (address) in a table using a Hash Function.
- Provides fast insert, delete and search.
- Used in Hash Tables, Dictionaries, Databases, etc.

- Applications :
- Caching
 - Symbol Tables
 - Database Indexing
 - Password Storage

Hash Table Example

Index	Value
0	-
1	Amit
2	-
3	Ravi
4	-
5	Neha
6	-
7	Soham

Hash Function Example :

$h(\text{key}) = \text{key} \% \text{table_size}$
table_size = 8

Key = 19

$h(19) = 19 \% 8 = 3$

So, index = 3

Collision Handling Techniques :

1. Separate Chaining
2. Open Addressing
(Linear Probing, Quadratic Probing, Double Hashing)



13. SYMBOL TABLE

- Symbol Table is an abstract data type (ADT).
- Stores **key-value** pairs.
- Supports operations: Insert, Delete, Search, Update.
- Used in Compilers, Interpreters, Dictionaries, Databases, etc.

Example Symbol Table (Student Roll Numbers)

Roll No.	Name
101	Amit
102	Ravi
103	Neha
104	Karan
105	Pooja

Operations :

- Insert(101, Amit)
- Search(103) → Neha
- Delete(102)
- Update(104, Mohit)

14. ALGORITHMS

- An algorithm is a finite sequence of well-defined instructions to solve a problem.
- Has **Input**, **Process** and **Output**.
- Must be Correct, Efficient and General.
- Characteristics of Algorithms :**

- Input
- Output
- Definiteness
- Finiteness
- Effectiveness

Example : Find Sum of First N Natural Numbers

Problem : Find the sum of first N natural numbers.

Input : N (a positive integer)

Output : Sum = 1 + 2 + 3 + ... + N

Algorithm :

Step 1 : Start

Step 2 : Read N

Step 3 : sum ← 0, i ← 1

Step 4 : While i ≤ N do
 sum ← sum + i
 i ← i + 1

Step 5 : Print sum

Step 6 : Stop

Example Run :

Input : N = 5

Sum = 1+2+3+4+5

Sum = 15

15. COMPLEXITY ANALYSIS

- Complexity analysis is used to measure the efficiency of an algorithm.
- Measured in terms of **Time** and **Space**.
- Helps to compare different algorithms.
- Asymptotic Notations** describe the upper bound of growth rate.

Common Time Complexities (Big-O Notation)

Notation	Name	Growth	Example
O(1)	Constant	1	Access array element
O(log n)	Logarithmic	log n	Binary Search
O(n)	Linear	n	Linear Search
O(n log n)	Linearithmic	n log n	Merge Sort, Quick Sort
O(n ²)	Quadratic	n ²	Bubble Sort, Selection Sort
O(2 ⁿ)	Exponential	2 ⁿ	Tower of Hanoi
O(n!)	Factorial	n!	Travelling Salesman

Space Complexity Example :

- If an algorithm uses fixed amount of memory regardless of input size → O(1)
- If it uses array of size n → O(n)

Why Important?

- To choose the best algorithm for a problem.
- To predict how an algorithm will perform as input size grows.
- To optimize time and memory usage.



16. RECURSION

- Recursion** is a technique where a function calls itself to solve smaller instances of the same problem.
- It has two parts :
 - Base Case (Stopping Condition)
 - Recursive Case (Function calls itself)
- Used in problems like Factorial, Fibonacci, Tower of Hanoi, DFS, Backtracking, etc.

Example : Factorial of N

Problem : Find N!

Definition :

$$N! = N \times (N-1) \times (N-2) \times \dots \times 2 \times 1$$

$$0! = 1$$

Recursive Algorithm :

```
fact(N):
if N == 0
    return 1    // Base Case
else
    return N * fact(N-1)    // Recursive Case
```

Example Run (N = 4)

$$\begin{aligned} \text{fact}(4) &= 4 \times \text{fact}(3) \\ &\downarrow \\ &= 4 \times (3 \times \text{fact}(2)) \\ &\downarrow \\ &= 4 \times (3 \times (2 \times \text{fact}(1))) \\ &\downarrow \\ &= 4 \times (3 \times (2 \times 1)) \\ &\downarrow \\ &= 24 \end{aligned}$$

Advantages :

- Simple Code
- Easy to Understand

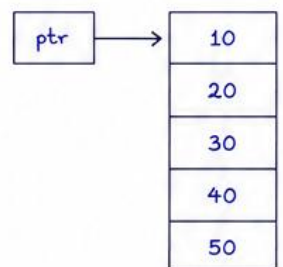
17. DYNAMIC MEMORY ALLOCATION

- In programming, memory can be allocated at **run time** using dynamic memory allocation.
- Done using **pointers** (in C/C++).
- Functions : malloc(), calloc(), realloc(), free()
- Provides **better memory utilization**.

Example (C Language)

```
int *ptr;
ptr = (int *) malloc(5 * sizeof(int));
// Memory for 5 integers
ptr[0] = 10;
ptr[1] = 20;
...
ptr[4] = 50;
free(ptr);    // Release memory
```

Memory Representation



Heap Memory

Functions Summary

Function	Use	Description
malloc(size)	Allocation	Allocates single block of memory of given size.
calloc(n, size)	Allocation	Allocates memory for n elements of given size and initializes all bytes to 0.
realloc(ptr, size)	Re-allocation	Changes the size of previously allocated memory block.
free(ptr)	De-allocation	Frees the allocated memory.

Advantages :

- Efficient Memory Usage
- Flexible Memory Size
- Reduces Memory Waste

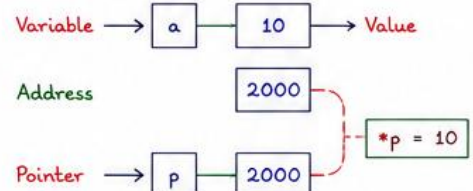
18. POINTERS (C/C++)

- Pointer is a variable that stores the **address** of another variable.
- Helps in dynamic memory allocation and efficient programming.
- Operators : & (address of), * (dereference)

Example (C Language)

```
int a = 10;
int *p;
p = &a;    // p stores address of a
printf("Address of a: %u", &a);
printf("Value of a: %d", a);
printf("Address stored in p: %u", p);
printf("Value at address p points to: %d", *p);
```

Memory Representation



Common Pointer Operations

- *p → value at address p
- &a → address of variable a
- p++ → moves to next memory location
- p-- → moves to previous memory location

Advantages

- Efficient Memory Access
- Used in Arrays, Linked Lists
- Supports Dynamic Memory

Note

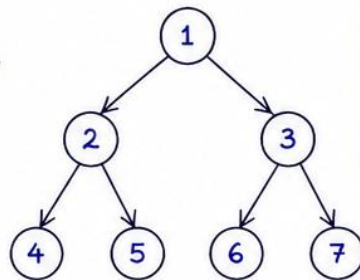
Pointers are powerful but must be used carefully to avoid memory leaks and dangling pointers.



19. TREE TRAVERSALS

- Tree Traversal means visiting each node of a tree **exactly once**.
- There are **three** common traversals.
- Used in searching, sorting, expression evaluation, etc.

Example Binary Tree



Types of Tree Traversal

Traversal	Order	Example (Above Tree)
1. Preorder	Root → Left → Right	1, 2, 4, 5, 3, 6, 7
2. Inorder	Left → Root → Right	4, 2, 5, 1, 6, 3, 7
3. Postorder	Left → Right → Root	4, 5, 2, 6, 7, 3, 1

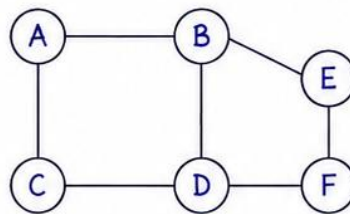
Applications :

- Inorder → gives sorted order of BST.
- Preorder → used to create a copy of tree.
- Postorder → used to delete the tree.

20. GRAPH TRAVERSALS

- Graph traversal means visiting **all vertices** of a graph.
- Two common traversals :
 - Breadth First Search (BFS)
 - Depth First Search (DFS)
- Used in shortest path, connectivity, cycle detection, etc.

Example Graph (Undirected)



BFS and DFS Example (Start from A)

Traversal Type	Visit Order
BFS (Level Order)	A, B, C, D, E, F
DFS (Depth First)	A, B, D, F, E, C

Applications :

- BFS → Shortest path in unweighted graph.
- DFS → Topological sort, cycle detection, connected components.

21. STACK APPLICATIONS

- Stack is used in many real-life applications.
- Based on **LIFO** (Last In First Out).
- Important applications are :

- Expression Evaluation
- Infix to Postfix Conversion
- Function Call Management (Recursion)
- Undo Operations
- Backtracking (e.g., N-Queens Problem)
- Browser History

Example : Infix to Postfix Conversion

Infix Expression : $A + B * (C - D) / E$

Postfix Expression : $A B C D - * E / +$

Algorithm (Using Stack) :

- Scan infix expression from left to right.
- If operand → add to output.
- If '(' → push to stack.
- If ')' → pop from stack to output until '('.
- If operator → pop from stack to output while top has higher or equal precedence.
- At last, pop remaining operators to output.

Advantages :

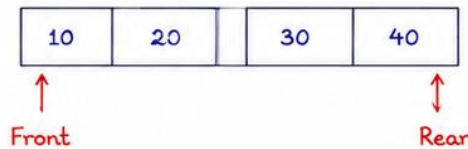
- Easy expression manipulation.
- Efficient memory usage.
- Useful in compiler design.



22. QUEUES

- Queue is a linear data structure.
- Follows **FIFO** (First In First Out)
- Insertion is done at **Rear**.
- Deletion is done at **Front**.
- Used in scheduling, buffering, printer queue, BFS, etc.

Example : Queue



Applications :

- CPU Scheduling (Round Robin)
- Printer Spooling
- Breadth First Search (BFS)
- Handling real-life waiting lines

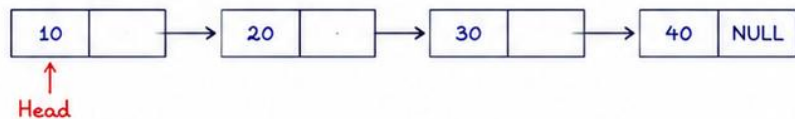
Basic Operations :

1. Enqueue (Insert)
2. Dequeue (Delete)
3. Front (Peek)
4. Rear (Peek)
5. isEmpty()
6. isFull()

23. LINKED LIST

- Linked List is a linear data structure.
- Elements are stored in **nodes**.
- Each node contains **data** and address of **next node**.
- Dynamic size (can grow or shrink).

Example : Singly Linked List



Types of Linked List :

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Circular Doubly Linked List

Applications :

- Dynamic Memory Allocation
- Implementing Stacks, Queues
- Polynomial Representation
- Sparse Matrix Representation

24. HASH TABLE

- Hash Table is a data structure that stores **key-value** pairs.
- Uses **Hash Function** to map key to index.
- Provides **fast** Insert, Delete and Search.
- Average Time Complexity : $O(1)$

Example : Hash Table (Hash Function : $h(k) = k \% 7$)

Index	Value
0	-
1	15
2	-
3	10, 17
4	4, 11
5	12, 19
6	6, 13

How it Works :

Keys : 10, 4, 15, 19, 17, 11, 12, 6, 13

$h(10) = 10 \% 7 = 3 \rightarrow$ index 3
 $h(4) = 4 \% 7 = 4 \rightarrow$ index 4
 $h(15) = 15 \% 7 = 1 \rightarrow$ index 1
 $h(19) = 19 \% 7 = 5 \rightarrow$ index 5
 $h(17) = 17 \% 7 = 3 \rightarrow$ index 3
 $h(11) = 11 \% 7 = 4 \rightarrow$ index 4
 $h(12) = 12 \% 7 = 5 \rightarrow$ index 5
 $h(6) = 6 \% 7 = 6 \rightarrow$ index 6
 $h(13) = 13 \% 7 = 6 \rightarrow$ index 6

Advantages :

- Very Fast Operations
- Efficient Searching
- Used in Dictionaries, Databases, Caching

Collision Handling :

1. Separate Chaining
2. Open Addressing



25. SORTING ALGORITHMS

- Sorting means arranging elements in a particular order (Ascending / Descending).
- Improves search efficiency.
- Common Sorting Algorithms :

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Heap Sort

Example (Ascending Order)

Unsorted Array :

64	25	12	22	11
----	----	----	----	----



Sorted Array :

11	12	22	25	64
----	----	----	----	----

Time Complexity Summary :

Algorithm	Best Case	Average Case	Worst Case	Space
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

26. SEARCHING ALGORITHMS

- Searching is the process of finding an element in a collection.
- Returns the position or presence of the element.
- Common Searching Algorithms :

- Linear Search
- Binary Search

(i) Linear Search

- Works on unsorted arrays.
- Checks element one by one.
- Time Complexity : $O(n)$

Example :

Array : 10 25 30 45 60
Key = 30
Comparisons : 10 → 25 → 30 (Found)
Position = 3 (Indexing from 0)

(ii) Binary Search

- Works on sorted arrays.
- Repeatedly divides the array into halves.
- Time Complexity : $O(\log n)$

Example :

Sorted Array : 10 20 30 40 50 60 70
Key = 40
Steps :
Mid = 30 → Key > 30 → Right Half
Mid = 50 → Key < 50 → Left Half
Mid = 40 → Found
Position = 3

27. TIME COMPLEXITY (BIG-O)

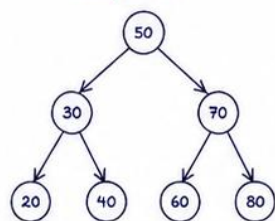
Common Time Complexities

- Time Complexity shows how running time increases with input size (n).
- Helps to compare efficiency of algorithms.
- Lower order is better.

Notation	Name	Growth	Example	What it Means
$O(1)$	Constant	1	Access array element	Time does not increase with n
$O(\log n)$	Logarithmic	$\log n$	Binary Search	Very slow growth
$O(n)$	Linear	n	Linear Search	Time increases linearly
$O(n \log n)$	Linearithmic	$n \log n$	Merge Sort, Quick Sort	Better than quadratic
$O(n^2)$	Quadratic	n^2	Bubble Sort, Selection Sort	Slower for large n
$O(2^n)$	Exponential	2^n	Tower of Hanoi	Very fast growth (bad)
$O(n!)$	Factorial	$n!$	Travelling Salesman	Worst growth (avoid)

28. BINARY SEARCH TREE (BST)

Example BST



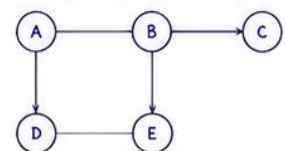
In-order : 20, 30, 40, 50, 60, 70, 80

- It is a Binary Tree.
- Left child < Root < Right child.
- Used for efficient searching, inserting and deleting.
- In-order traversal gives sorted order.

29. GRAPH (BASICS)

- Graph is a collection of vertices (nodes) and edges.
- Types :
 - Directed Graph
 - Undirected Graph
- Represented using Adjacency Matrix or Adjacency List.

Example Undirected Graph



Applications :

- Social Networks
- Network Routing
- Maps & Navigation
- Dependency Graphs



MARUDHARA EXAM

आपकी सफलता, हमारा संकल्प

www.marudharaexam.in

Page 9 of 10

(2 Pages Remaining)

DATA STRUCTURES AND ALGORITHMS

Page - 10

★ FINAL REVISION SHEET - MOST IMPORTANT ★

1. IMPORTANT DEFINITIONS

- **Data Structure** is a way of organizing and storing data so that it can be used efficiently.
- **Algorithm** is a finite set of well-defined steps to solve a problem.
- Data Structures provide the best way to **store** and **access** data.
- Algorithm provides the logic to process the data.

2. DATA STRUCTURES QUICK VIEW

Data Structure	Key Points / Use
Array	Stores elements of same type in contiguous memory.
Linked List	Collection of nodes, dynamic size.
Stack	LIFO (Last In First Out).
Queue	FIFO (First In First Out).
Tree	Hierarchical structure (Root → Leaves).
Graph	Collection of nodes and edges.
Hash Table	Stores key-value pairs, fast access.

3. COMPLEXITY CHEAT SHEET

Operation	Array	Linked List	Stack	Queue	BST (Avg.)
Access / Peek	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(\log n)$
Search	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$
Insert	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$
Delete	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$

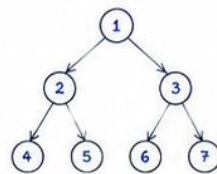
IMPORTANT PROPERTIES

- **Time Efficiency** → Less time taken
- **Space Efficiency** → Less memory used
- **Scalability** → Handle large data
- **Reliability** → Accurate and correct

4. ALGORITHMS PARADIGMS

1. Brute Force
2. Divide and Conquer
3. Greedy Method
4. Dynamic Programming
5. Backtracking

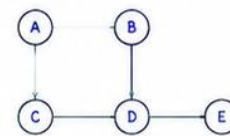
5. TREE QUICK RECAP



Types of Tree :

- Binary Tree
- Binary Search Tree
- AVL Tree
- Heap (Min / Max)
- B-Tree

6. GRAPH QUICK RECAP



Types :

1. Directed Graph
2. Undirected Graph

Traversals :

- BFS (Level Order)
- DFS (Depth First)

7. IMPORTANTS FORMULAS & NOTES

- **Factorial** : $n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$
- **Sum of first n natural numbers** : $\frac{n(n+1)}{2}$
- **Sum of squares** : $\frac{n(n+1)(2n+1)}{6}$
- **Fibonacci Series** : 0, 1, 1, 2, 3, 5, 8, ...
- **Binary Tree Max Nodes at Level i (0-indexed)** : 2^i
- **Total Nodes in Full Binary Tree (height h)** : $2^{h+1} - 1$
- **Height of Full Binary Tree with N nodes** : $h \approx \log_2(N+1) - 1$

8. MUST REMEMBER POINTS

- **Array** has fixed size, **Linked List** is dynamic.
- **Stack** → use push() and pop() (LIFO)
- **Queue** → use enqueue() and dequeue() (FIFO)
- **BST** → Left < Root < Right
- **BFS** → uses Queue, **DFS** → uses Stack / Recursion
- **Hash Table** → fast search, insert, delete
- **Big-O** shows **worst case** growth
- **Always** handle boundary conditions in code

9. COMMON COMPLEXITIES MEMORY TRICK

$O(1)$	→ Constant	"1 log n nlogn n* 2^n n! Always Remember This Order"
$O(\log n)$	→ Logarithmic	
$O(n)$	→ Linear	
$O(n \log n)$	→ Linearithmic	
$O(n^2)$	→ Quadratic	
$O(2^n)$	→ Exponential	
$O(n!)$	→ Factorial	

10. PYQ TYPE MCQS (PRACTICE)

1. Stack follows _____ order.
(A) FIFO
(B) LIFO
(C) Sorted
(D) Random [Ans: B]
2. In binary search, array must be _____.
(A) Sorted
(B) Unsorted
(C) Both
(D) None [Ans: A]
3. Time complexity of binary search is:
(A) $O(n)$
(B) $O(\log n)$
(C) $O(n^2)$
(D) $O(1)$ [Ans: B]
4. In BST, left subtree of a node contains values _____. root.
(A) Greater than
(B) Less than
(C) Equal to
(D) None of these [Ans: B]
5. Which data structure uses FIFO?
(A) Stack
(B) Queue
(C) Tree
(D) Graph [Ans: B]
6. Which sorting algorithm has best average case?
(A) Bubble Sort
(B) Merge Sort
(C) Insertion Sort
(D) Selection Sort [Ans: B]
7. Hashing is used for _____.
(A) Sorting
(B) Fast Searching
(C) Graph Traversal
(D) All of these [Ans: B]
8. Space complexity of iterative binary search is:
(A) $O(1)$
(B) $O(\log n)$
(C) $O(n)$
(D) $O(n \log n)$ [Ans: A]

EXAM SUCCESS TIPS

- ★ Understand concepts clearly.
 - ★ Practice more questions.
 - ★ Analyze time complexities.
 - ★ Write clean and optimized code.
 - ★ Stay calm and confident in exam.
- ALL THE BEST ! 😊



MARUDHARA EXAM

आपकी सफलता, हमारा संकल्प

www.marudharaexam.in

Page 10 of 10

(1 Pages Remaining)